情報処理の主要問題論説

I　アルゴリズムとその解析の動向と展望†

福　村　晃　夫††

1. はしがき

ちかごろアルゴリズムという言葉がしきりに使われるようになって，そこに時代の移り変わりがあるように思われる．1950年ごろ，Information Theory がわが国にも導入されたとき，この理論を“情報理論”と呼ぶのにはかなりの抵抗感があった．その後しばらくして，この言葉は情報の研究分野にすっかり定着したが，研究の内容や成果を情報科学，情報工学の呼び名で分野外に披露することには，なにか面映ゆい感じがしたことであった．そのような言葉がいまは街に氾濫している．情報という日常用語が学術用語となり，それがまた，別の意味を持って社会に還元されたといえるであろう．

アルゴリズムは，計算の，もっと広く言えば情報処理の真髄であり，研究者や技術者の間では使い馴らされている言葉である．ちょうど過去の情報の場合と同じように，今度はこの言葉が，計算ないし計算機という重大な意味を背負って，社会に流布されようとしている．アルゴリズムの必要性と重要性に対する幅広い認識が急速に深まり，工芸的あるいは研究的興味の対象と見なされがちであったアルゴリズムが，系統だてられた技術として機能し始めるのも間もないことと予想されるのである．

さて，本学会が創立されて間もないころ，筆者の傍には Bellman の Dynamic Programming[1] の書物があった．この本の著者は，次元性の呪 (curse of dimensionality) という言い方をして，むかしからこの呪によって多くの問題が解けないでいるが，本質的な問題は，次元性の克服よりはむしろモデリングにあるとして，自己のモデルを紹介している（同書序文）．しかし，そのモデルを用いてもやはり呪から逃がれる工夫をせざるを得ないことを，別の書物[2]で述べている．

具体的な例としては，m個の制約条件 $\sum_{j=1}^{n} b_{ij}x_j = c_i\ (i=1,2,\cdots,m)$ のもとで，目的関数 $\sum_{j=1}^{n} g_j(x_j)$ を最大にする最適化問題があげられている．よく知られているように，動的計画法では，目的関数の最大値を f_n として，漸化式

$$f_n(c_1, c_2, \cdots, c_m) = \max_{x_n} [g_n(x_n) + f_{n-1}(c_1 - b_{1n}x_n, c_2 - b_{2n}x_n, \cdots, c_m - b_{mn}x_n)] \quad (1)$$

を，$n=1,2,\cdots,n$ について計算する．f_i を計算するときに，$i>k$ である f_k の再計算は一切不要であることがこの方法の利点であるが，もし各 f_i に対する各初期値 c_j をK個に刻むとすると，計算回数は nK^m となる．K^m が呪われた数であることは直ちに分かることである．

ところで，その後 20 年の歳月を経て，果してどれだけの問題がこの呪から解放されたであろうか．以下，このことから史実を拾い上げてみることにするが，その前に，アルゴリズムの問題の原点にもどるのが至当であろう．

2. Church の提唱

アルゴリズムの起源については，われわれの立場からすれば，Church の提唱までたどれば十分であろう．それは，

「計算可能な関数とはチューリング機械によって計算される関数のことであり，もし計算の過程が停止すれば，計算のアルゴリズムが存在する」

という趣旨のものである．

この提唱によると，アルゴリズムが存在するかどうかを決めるには，チューリング機械（TM と略記する）の計算が停止するかどうかが決定できなくてはならない．これが有名な TM の停止問題であって，そ

† A Trend and a View of Algorithms and Thbeir Analysis by Teruo FUKUMURA (Department of Information Science, Nagoya University).

†† 名古屋大学工学部情報工学科

の否定的な解答の論証に使われた対角化論法とシミュレーションは，アルゴリズムに関する重要な理論的結果を得るときも有効に使われた．そこで，論証の方法を簡単に説明しておこう．

TMのモデルは多くの教科書でなじみ深いであろう．それは，テープに対する読み書き用のヘッドを持つ，内部状態数が有限の抽象的な装置である．形式的には，TMはつぎの4項組で表わされる．

$$(q_i, s_j, s_k, q_l),\ (q_i, s_j, R, q_l),\ (q_i, s_j, L, q_l)$$

いずれも，装置がq_iの状態にあるとき記号s_jを読んで状態をq_lに変えることを意味しており，第1のものは，次の動作として記号s_kを印刷する，第2，第3のものは，印刷はしないで，ヘッドをそれぞれ右，左へ移動させることを意味する．これらは計算機の命令に相当しており，TMは，プログラムにあたる有限個の4項組の集合として定義される．これをZとしよう．Zは記号の系列で表わされる．

記号系列Xを入力とする任意のTM Zの計算を(Z, X)で表わし，この計算をシミュレートするTM Z_0を考える．Z_0への入力は記号系列ZXであり，Z_0はテープ上のZを読みながらXを書き換えていく．とくに，Z_0への入力をZZとする（対角化する）と，Z_0は，任意の計算(Z, Z)をシミュレートすることになる．

いま，Z_0の停止問題を解くTM（あるいはアルゴリズム）Hが有るとしよう．それは，Zを入力されてZ_0の停止，非停止を判断する．そこで，Hに4項組を追加して，次のようなTM H'をつくる．すなわち，H'はHと同じ入力を与えられるが，HがZ_0は停止すると判断するとき永久ループに入り，Z_0は停止しないと判断するとき停止するようにする．ここで，H'への入力をH'としてみる．Z_0は計算(H', H')をシミュレートし，H'がZ_0は停止しないとして停止するとき停止し，H'がZ_0は停止するとして停止しないとき停止しない．これは矛盾であるから，TMの停止問題を解くアルゴリズムは存在しない．

上記の問題を契機にして，アルゴリズムの存在を問う多くの決定問題が研究されたが，それらの研究においては，計算に要する資材を考慮する必要は全くない．これに対して，計算に要する空間（記憶）量，時間量の概念を導入して，アルゴリズムが存在する場合の計算の複雑さ（complexity）を解析する研究が盛んに行われた．1960年代後半から70年初頭にかけて急速に進展した形式言語理論の流れを汲むもので，TMによる言語（入力の集合）の認識の複雑さの評価が行われた．

得られた成果には，加速定理とかテープ節約定理といった興味深いものが多いが，それらのほとんどは，アルゴリズムの具体的な内容は規定せず，空間量，時間量の漸近的な振舞いを追究するものである．したがって，この話題に関しては，問題の複雑さに関する階層性を示したHartmanisらの定理[3]のみを次章で示すことにしよう．

3. 多項式的と指数関数的

アルゴリズムや問題の複雑さを評価するのに，多項式的（nを変数，c_1, c_2を定数として$c_1 n^{c_2}$と表わされる）と指数関数的（$c_1 c_2^n$と表わされる）との間で確然とした差別をつけるべきであることをはじめて明確に主張したのは，1965年のEdmondsの論文[4]ではないかといわれる[5]．この論文はグラフの問題に関するものであって，本論に入る前の若干のわき道（Digression）で，著者は，「――実際上の目的には，オーダが代数的であるか指数関数的であるかの異なりが，有限か無限かの差異より重要であることはしばしばあるであろう」と述べている．

ところで，n個の変数を持つある難問が与えられたとき，これを解くにはc^nの時間量を要するアルゴリズムしか無いのか，という疑問が持たれるであろう．前述のHartmanisらの定理は，そのような問題があることを示唆している．いま，TMへの入力（記号系列）の長さをn，nのある関数を$S(n)$とするとき，TMの計算が終了するまでに費やされるテープ量（マス目の数）がたかだか$S(n)$であれば，このTMの空間的複雑さは$S(n)$であるという．（時間的複雑さも同様に定義される．）定理は次のようである．

「関数$S_1(n) \geqq n$，$S_2(n) \geqq n$を空間的複雑さの関数とするとき，もし，

$$\inf_{n\to\infty} \frac{S_1(n)}{S_2(n)} = 0 \qquad (2)$$

であれば，複雑さが$S_2(n)$のTMでは受理されるが，$S_1(n)$の複雑さのTMでは受理されない言語が存在する．」

ここに，infは最大の下界である．厳密には，TMは決定性といわれるものでなくてはならないが，その定義は後に行う．

この定理の証明にも既出の対角化論法とシミュレーションが用いられる．そのごく大要を説明すると次の

ようである．任意の TM をZとし，その入力がZであるときの計算をシミュレートする TM M_0 をつくる．M_0はZを入力され，Zがテープ量$S_1(|Z|)$（$|Z|$はZの長さ）以内でZを受理すれば受理をせず，それ以外の場合は受理するようにつくられる．M_0 の複雑さは $S_1(|Z|)$ であるとして，Zを M_0 に置き換えると矛盾が生じる．

式(2)を導くのも手数がかかるので省略するが，M_0 の構成手順は具体的に示されるので，M_0 で受理されるある言語 L が存在し，L は複雑さが $S_1(n)$ の TM Z では受理できないことになる．したがって，複雑さが 2^n の TM（アルゴリズム）では受理（処理）されるが，$2^n/n$ の複雑さでは処理できない言語（問題）が存在する可能性がある．

この定理を利用して，本質的に複雑な問題を実際に示すことが行われた．ここでは，実際的と考えられる属性文法 (attribute grammar) が本質的に手に負えない (intractable) 問題であることにふれておく[6]．

属性文法は，Knuth[7] によって提案された文法であり，その基本は文脈自由形文法であるが，生成される言語のもつ意味 (semantics) を記号の属性を介して構文にともなわせる工夫がほどこされている．文献[6]に用いられている例で説明すると，記号の属性があらかじめ，

synthesilzed : $\{a_1, b_1, c_1\}$

inherited : $\{a_2, b_2, c_2, c_3\}$

のように定められているとする．これらの属性は，具体的には，非終端記号のデータの形であったり，数値であったり，記憶場所であったりする．またこれらは，生成規則が $X \to \beta$ であるとき，生成によって合成される X の属性，および β の中の非終端記号に受けつがれる属性を表わすように用いられる．

これらの属性を用いて，生成規則に次のような意味規則 (semantic rule) をともなわせる．

1. $A \to BCx$

   $a_1 = b_1 + 5, \quad c_2 = 26 * c_1$

   $b_2 = f(c_1), \quad c_3 = 56$

2. $B \to Bx$

   $(b_1)^l = (b_1)^r, \quad (b_2)^r = (b_2)^l / 124$

3. $B \to x$

   $b_1 = b_2$

4. $C \to y$

   $c_1 = 4$

ただし，2. で用いられている添字 l, r は，生成規則の

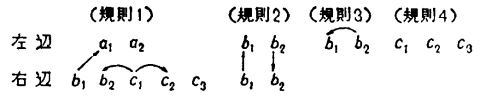


図-1 生成規則の適用による属性の決まり方

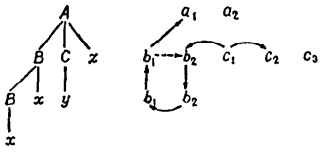


図-2 構文木と，それに対応する属性の依存関係

左辺，右辺の属性であることを意味する．また，f は既知の関数である．

各生成規則の適用による属性の決まり方を図示すると，図-1 のようになる．たとえば $b_1 \to a_1$ は，a_1 の前に b_1 が評価されるべきことを示す．簡単な生成例と，そのときの構文木にともなう属性間の依存関係を示すと，図-2(a), (b)のようである．もし，図-2(b)に点線で示される矢印があるとすると，属性の評価に矛盾が生じることがわかるであろう．文法にこのような巡回性があるかを決める問題の複雑さが，文法の大きさとともに指数関数的に増大せざるを得ないことが証明されるのである．

一般に，問題を解く手順が木の探索で表わされるとき，たとえ木の高さは問題の大きさの多項式で与えられるとしても，探索にバックトラックがともなうと，手順は急激に複雑になる．文脈自由形文法は比較的単純であり，それに関する問題で多項式の時間で解けるものは多いが，意味がかかわると問題がいちじるしく複雑になることを，この例は示している．意味情報の構造化が困難であるゆえんである．

4. $\mathcal{P}$ と $\mathcal{NP}$

この章では，まず非決定性 TM (nondeterministic TM, NDTMと略記する）の説明をしなければならない．TMの4項組において，q_i と s_j の1つの組に対する TM の動きが一意に定まるとき，TM は決定性 (deterministic) TM (DTM と略記する）と言われる．これに対し，NDTM では，1つの q_i, s_j の組に対する動きが何とおりも許される．したがって，NDTM では，いくとおりもの計算が並列に進行し，それはどれだけ長く続いてもよい．ゆえに非決定性なのである．

DTM による言語の認識（受理）問題が問題の大きさの多項式で与えられる時間（多項式時間という）で解かれるとき，この問題はクラス $\mathcal{P}$ (polynomial) に属すると言われる．これを $\mathcal{P}$ 問題といおう．一方，NDTM によって並列に行われる計算のいずれか一つが多項式時間で結果を出せば，この問題はクラス $\mathcal{NP}$ (nondeterministic polynomial) に属すると言われる．これを $\mathcal{NP}$ 問題といおう．同様の定義は，空間量に対しても行われている．$\mathcal{NP}$ 問題については次の注意が必要である．

NDTM によって並列的に行われる計算の個々は決定性である．いま，入力は受理されるべきものであるとしよう．いくとおりかの計算の過程の中には必ず受理に到達するものが存在する．したがって，その1つが示されれば，その過程を決定性の手続きで追うことによって，受理を決めることができる．そのときの時間が多項式時間であるから $\mathcal{NP}$ であるといわれるのである．ところが，入力が受理されるべきものでなければ，どの計算も肯定的な答を出さないから，受理されないことを多項式時間で確かめる決定性の手続きは無いかも知れない．すなわち，ある $\mathcal{NP}$ 問題を否定して得られる問題は，必ずしも $\mathcal{NP}$ 問題ではない．

$\mathcal{NP}$ 問題が研究者の衆目を集めたのは，古くから呪われたアルゴリズムしかなくて，札つきの難問とされて来た巡回セールスマンの問題をはじめとする数々の問題が，言語認識の問題に帰着させると $\mathcal{NP}$ のクラスに属するからである．$\mathcal{P}\subseteq\mathcal{NP}$ であることは，それぞれの定義から明らかであるが，もし，$\mathcal{P}=\mathcal{NP}$ が証明されれば，次元性の呪は霧散するであろう．

この問題の解明で中心的役割を果しているのは，1971年に Cook[8] によって示された $\mathcal{NP}$-完全 ($\mathcal{NP}$-complete) と呼ばれる問題のクラスである．このクラスに属する問題は，そのいずれか1つが $\mathcal{P}$ 問題であることが分かれば，他のすべての $\mathcal{NP}$ 問題も $\mathcal{P}$ 問題であることになる，という意味で，きわめて難しい問題である．しかし，はじめて $\mathcal{NP}$-完全であることが示された問題は，ブール式の充足可能性 (satisfiability) といわれるきわめて単純な問題である．すなわち，与えられたブール式の値を真にする論理変数の組合わせは存在するか，という問題である．変数の数を m とすると，2^m とおりのすべての場合について式の真理値を計算するまずい方法なら誰でもが思い付く．しかし，これより格段に良い方法はまだ知られていない．

この問題が $\mathcal{NP}$ であることは容易に分かる．たとえば，ブール式は

$$(p_1\vee p_2)\wedge p_3$$

であるとしよう．非決定性の考えで複雑さを評価するのであるから，解 $p_1=1$, $p_2=0$, $p_3=1$ を1つ与えればよい．式を左から走査して，これらの値を1つ1つ代入していく初歩的な方法でも，必要なステップ数は悪くて n^2 である．ここに，n は式の長さである．

この問題が $\mathcal{NP}$-完全であることを論証するには，任意の NDTM による言語の認識問題が，多項式時間でブール式の充足可能性の問題に帰着されることを証明する．さきに 2 章で示したように，TM は4項組の集合で定義されたが，各4項組は TM の動きを規定する規則にほかならない．したがって，それらから「——は規則を満たしているか」という TM の動作に関する論理命題を導くことができる．一方，テープ上のヘッドの位置と記号系列（計算の初期は入力）は，計算のステップごとに変化していくが，TM は必ず停止する（$\mathcal{NP}$ 問題ではこの場合だけを考えればよい）から，テープ上におけるそれらのパターンは有限個しかない．よって，ある入力が TM によって受理されるかという問題は，テープ上に生じうるパターンの有限個の系列のうち，論理命題を満たすものがあるかという問題に変換される．しかも，この変換が多項式時間で終了することも証明されるのである．

$\mathcal{NP}$-完全な問題には，数多くのものがある．それらを列挙する余裕は無いので，たとえば Aho らの著書[9]の 10 章，およびその演習問題に詳しいことだけを記しておく．

いく多の英知の挑戦にもかかわらず，$\mathcal{P}=\mathcal{NP}$? の問題はいまだに解決していない．したがって，この問題は現在でもアルゴリズムの理論的研究の中心的課題であるが，一方で，これだけの努力にもかかわらず，という理由で，問題の否定的解答を予想する傾向もある．ここ 20 年間で，問題の複雑さについてはずいぶん多くのことが分ったが，いまだに複雑さの壁を破れずにいる問題が数多く残されている，というのが現状であろう．

5. 多項式時間で解ける問題

前章までの理論研究とは別に，多項式時間で解けるあつかい易い (tractable) 問題と，そのアルゴリズムについても，おびただしい研究と経験の蓄積がある．本章ではそれらをまとめてみよう．

5.1 アルゴリズムの設計法

アルゴリズムの設計にも，組織的な方法論とよく整備された道具を備えるために，工学が存在することが望ましい．Aho ら[9]を参照すると，従来から使われて来た諸手法は次のように整理されるが，これも一つの工学であると言えよう．

（1） データ構造　アルゴリズムの良さは使用するデータ構造に強く依存する．その基本となるものは配列とリンク構造であり，それらに対する基本操作は蓄積と検索である．基本構造を利用して，リスト，集合，グラフ，木などの構造が構成される．基本操作は，リストに対しては追加，削除など，集合に対しては find, union など (5.3 参照)，グラフ，木に対しては，depth-first-search (DFS), breadth-first-search (BFS) などの基本的探索手順がある．

（2） 再帰的方法 (recursion)　集合 S の要素を大小順に並べるための Hoare のアルゴリズム Quicksort は，次のように記述される．

このアルゴリズムを Quicksort (S) とする．

Step 1. Sの要素数が1であればSを出力する．

Step 2. Sから要素xを選び，Sを，xより小さな要素の部分集合 S_1, x を要素とする部分集合 S_2, x より大きな要素の部分集合 S_3 に分割する．

Step 3. Quicksort (S_1), S_2, Quicksort (S_3) を出力する．

Step 1, Step 2 におけるアルゴリズムの定義が，Step 3 で再び用いられている理由で，再帰的な記述法であるといわれる．この方法によると，アルゴリズムはきわめて簡潔に表現されるし，分り易い．

（3） divide-and-conquer と balancing　要素数が n の集合Sの中から最大値を探すとき，まず，Sを要素数が $n/2$ の部分集合に分割し，それぞれの最大値を比較して全体の最大値を求めることを考える．同じ考えを再び部分集合に適用することを考えると，最大値を求めるアルゴリズムの再帰的な表現に思いつくであろう．このような方法が divide-and-conquer と言われ，所要ステップ数を減少させる要因となる．集合を2等分するのが balancing であり，これもアルゴリズムの効率化に有効である．

（4） 動的計画法　これは 1 章で説明した．基本的な考え方は再帰性にあることが分かるであろう．

5.2 アルゴリズムの解析法

設計には設計結果の解析がつきものである．これについてまとめてみよう[10]．

（1） 計算機モデル　複雑さの解析に実用電算機の動作を持ち込むのは複雑すぎるし，TM はテープ作業をともなう特別の問題以外には単純すぎるという理由で，RAM (random access machine) と呼ばれるモデルがよく使用される．RAM と TM による複雑さの評価の間には，多項式で与えられる差異しかない[9]．

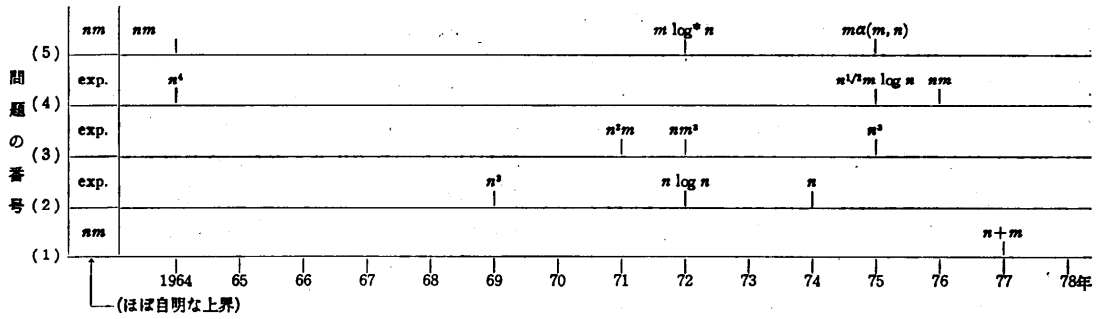
（2） 複雑さの尺度　アルゴリズムにおける主要演算のステップ数，レジスタへのアクセス回数など，時間尺度が用いられることが多い．命令によらず，実行回数を一様に計数する方法もある．また，アクセスする情報を記述するに必要なビット数を問題にする対数的コストの規準[9]もある．しかし，どの尺度を用いても複雑さのオーダには変わりはない．

（3） 最悪の評価と平均的評価　入力の条件が最悪の場合について複雑さの評価が行われることが最も多い．解析の容易さが主な理由である．しかし，実際の問題では平均的な入力が圧倒的に多いので平均的評価が重要視されている．確率分布が推定し難い，分布の仮定が現実的でない，問題の解析が難しく，とくに分散が評価できない，などがこの方法の難点であり，今後の課題である．

（4） 下界と上界　複雑さの絶対値の評価は難しいので，その上，下界がよく評価される．問題の大きさ（入力の個数など）を n，複雑さの関数を $g(n)$ として，$g(n)$ の成長が既知関数 $f(n)$ より速くなければ $g(n)=O(f(n))$ と書いて上界を表わし，遅くなければ $g(n)=\Omega(f(n))$ と書いて下界を表わす．

問題の複雑さの下界を求めることは難しい．理由は，どのようなアルゴリズムにも最低必要な時間量を評価する必要があるからである．自明な下界の求め方に，入力数を n，出力数を m として，$\Omega(n+m)$ とする方法がある．自明であるとはいえ，これより良い下界はなかなか求まらないのである．情報理論的方法といわれる方法もときに有効である．たとえば，n 個の要素の中の1個を探索するには，アルゴリズムに関係なく，$\log_2 n$ の手数が必要である．この方法は，入力に対して無知を装うことから生まれるが，他に，どのようなアルゴリズムでも手古ずるような“意地の悪い”場合を考えて，下界を導く方法もある．

上界としては，与えられた問題に対して，現在わかっている最高速のアルゴリズムの時間量の上界をとればよい．したがって，上界を探すことと，より速いアルゴリズムを開発することは等価である．最悪の入力条件を求めること自体にも問題があるが，それが求まった後の評価には，命令の実行回数を直接計数する方



(注) (1) $\log^* n = \min(i \mid \underbrace{\log \log \cdots \log}_{i\text{回}}\, n \leq 1)$

(2) Ackerman 関数を $A(i,j)$, 数 x 以下の最大の整数を $\lfloor x \rfloor$ とするとき,

$\alpha(m,n) = \min(i \geq 1 \mid A(i, \lfloor 2m/n \rfloor) \geq \log_2 n)$

とくに, 関数 α の成長はきわめて緩やかである.

図-3 各種問題の複雑さに対する上界の改良

法と, 実行回数に関する差分方程式を導いて, それを解く方法[9]とがある.

5.3 上界の向上

上界については多くのことがわかっており, 組合せ的問題に対しては, Tarjan のすぐれたサーベーがある[5]. そこから資料を借りて, ごく限られた例ではあるがアルゴリズムの改良の年表をつくり, 図-3 に示す. 以下に, ①問題と内容, ②問題の大きさのパラメータ, ③用いられた手法を示す.

(1) ①パターンマッチング——記号系列から指定された部分系列を抽出する, ②記号系列と部分系列の長さ n, m, ③データ構造, オートマトン理論.

(2) ①グラフの平面性——辺の交差なしにグラフが平面上に書けるかの検査, ②頂点数 n, 辺数 m, ③DFS.

(3) ①最大ネットワークフロー——辺に流量の制約のある有向グラフにおいて, source からの最大流量を求める, ②頂点数 n, 辺数 m, ③BFS, augmentation (逐次的最適化手法)

(4) ①グラフマッチング——無向グラフ中の, 互いに頂点を共有しない辺の最大数を求める, ②頂点数 n, 辺数 m, ③BFS, cycle shrinking (グラフの操作), augmentation.

(5) ①集合和——集合 $S_1, S_2, \cdots, S_n$ がそれぞれ相異なる1個の要素からなるとき, それらに対し, 操作 find (x) : x を与えられて, x を含む集合の名前を求める, union (A, B, C) : 名前が A, B の集合の和を求めて名前を C とする, を繰り返し用いて全集合の和をとる. (最小コストの張木 (spanning tree) を求めるときなどの基本演算である.) ②union の回数 n, find の回数 m, ③path compression (データ構造).

図-3 を見ると, 過去10年間に著しい進展があったことがうかがわれる. 中でも, 問題(1), (2) の上界は線形であるから, これらは下界ともみなされる. また, 問題(5)の最近の上界は, 下界でもあることがわかっている. 一般に, 非線形の下界は求め難いものであるから, この結果はきわめて興味深い. 問題 (1) は, アルゴリズムを手広く研究している Knuth が, はじめてオートマトン理論に助けられたことを述懐している, 示唆的な事例である[11].

5.4 近似アルゴリズム

いかに指数関数的時間量を要するとはいえ, そのような問題を解く立場に立たされるのが現実である. まして $\mathcal{P}=\mathcal{NP}$? の問題に対して否定的解答が予想される事態では, 手に負えない問題のアルゴリズムの効率化が工夫されなければならない.

組合せ的な問題の平均的計算量を減少させる方法として, 分枝限定法と呼ばれる方法がある. 問題が 0, 1 計画法であるならば, アルゴリズムは2分木の探索で表現される. 木の水準 i の節点は, たとえば, 変数 $x_1, x_2, \cdots, x_i$ の値の1つの組合せ (部分解) に対応する. さらに残りの変数 $x_{i+1}, \cdots, x_n$ に値0を代入したとき, もし制約条件が満たされなければ, 水準 i におけるその節点は可能解にはなり得ない. よって, その節点からの分枝はもはや行わない. これがこの方法の特徴であり, さらに, 可能解の候補のどれから分枝するかについて, ヒューリスティックが用いられる. しかし, そのようにしても, 最悪な場合, 計算量はやはり指数関数的になり, 次元性の呪はさけられない.

ある種の近似を許すことによって多項式時間で解を

得ようとする方法は，大きく分けて二つある．その一つは，得られる解はつねに近似解であるが，相対誤差は一定値以内であることが保証される方法である．他の一つは，ほとんどの場合は真の解が得られるという，確率的な方法である．

さきの 0, 1 計画法を前者の方法で解くには次のようにする[10]．水準 i の各節点の部分解を用いて，節点ごとに，目的関数の部分値（部分的に評価された値）f_i を求め，その最大値を F_i とする．また同じようにして，制約条件式の部分値 g_i も求める．次に，実数 ε を $0<\varepsilon<1$ のように定め，区間 $[0, F_i]$ を幅 $F_i\varepsilon/n$ の部分区間に分割する．水準 i の全節点を f_i によって各部分区間に分類し，同じ類に属する節点のうち，g_i の最小のもの（最大問題の場合）のみを残し，他は無視する．

上記の手順をふめば，各水準で生き残る節点の数はたかだか n/ε であり，木の高さは n であるから，複雑さは $O(n^2/\varepsilon)$ となる．一方，各水準で生じる誤差は ε で，加法的であるから，アルゴリズム全体の誤差は ε のオーダである．

しかし，この種の方法がどのような $\mathcal{NP}$ 問題に対しても有効であるわけではない．“もし，ある相対誤差のもとで多項式時間のアルゴリズムが構成されるならば，任意の相対誤差に対する多項式時間のアルゴリズムが存在する”ことが証明されている問題もある（グラフの最大クリーク問題）．

一方，確率的な方法は一種のモンテカルロ法であるといえよう．たとえば[12]，奇整数 n が素数であるかどうかを決定する（$\mathcal{NP}$-完全ではないが $\mathcal{NP}$ 問題である）のに，一様乱数 $1, 2, \cdots, n-1$ の中から整数 a を 1 個選び，n と a が互いに素であれば，剰余 $r \equiv a^{(n-2)/2} \pmod n$ $(-1 \leqq r < n-1)$ と，Jacobi の記号 $\delta = (a/n)$（説明は省略する）を計算する．もし，$r=\delta$ であれば n は素数と決定する．もし，$\gcd(a, n)>1$ であるか，$r \neq \delta$ であれば，n は合成数と決定する．この手順によると，n が素数のときは必ず正解が得られ，n が合成数のときは，m 回の試行がもたらす誤り確率は 2^{-m} であり，そのときの算術演算の回数は $6m \log_2 n$ であることが示される．

この種の方法も最近活発に研究されているようであるが，確率事象の利用にまつわる諸々の難点の克服が欠かせないことは，従来のモンテカルロ法と同じである．

6. むすび

チューリング機械の停止問題から始まったアルゴリズムの話題は，最後には確率的近似法という異質とも思われるものまでに及んだが，それほどまでに，アルゴリズムと複雑さの問題は幅広く，奥深いようである．それらをいま一度べっ見して，本文を締めくくることにしよう．

複雑さの理論研究では $\mathcal{P}=\mathcal{NP}$？の問題が中心的課題であり，その周辺の問題を含めて綿密な研究が今後も続けられるであろう．その解答がどう出るかは予断を許さないが，アルゴリズムの理論研究における重要な成果のほとんどが，対角化論法とシミュレーションに依存していることに着目すると，論証の手法として，それらとは異なる新しい（離散的）数学の概念が導入されることが期待される．また，DTM と NDTM との間を埋める認識手法の提案も期待される．

多項式時間の複雑さの上，下界は，今後も改良されていくであろう．それらの評価が，問題の大きさ n の値の大きい場合について行われているのは，最近の電算機の高速化によって正当化されるという説[9]もある．しかし，小さな n の値に対する複雑さの定数係数の効果を考慮して，アルゴリズムを比較することも重要である．また，空間量，時間量の複雑さの相互関係を明確にする必要がある．

最適化の一手法として有名なシンプレックスアルゴリズムの計算量は，最悪の場合，問題の大きさとともに指数関数的に増大することがわかっている．しかし，実際にそのような計算量を経験した人はあまりあるまい．その意味で，複雑さの平均的評価の手法の研究は積極的に進められるべきである．また，手に負えない問題に対する近似アルゴリズムの開発も重要課題である．

最後に，本文ではふれなかったが，実際的な並列アルゴリズムの提案とその効率の評価法を，とくに今後の重要課題として指摘したい．

さて，計算あるいは処理にアルゴリズムがともなうことは自明である．この自明さが，アルゴリズムの一般性，大衆性につながるところに情報処理の時代的な様相がうかがわれる．しかも，アルゴリズムは規則の集団であって，みじんのあいまいさも許されない．その設計，解析に関する系統立てられた技術が，アルゴリズム研究の進展の波面として，その表現であるソフトウェアの技術ともども社会に伝わるのが 1980 年代

であると思われるし，また，そのようであることを期待したい．

なお，本文で参照されている文献は，アルゴリズムの遍歴における一里塚ないし見晴らし台とみなされるもののみであることをことわっておく．

参考文献

1) Bellman, R.: Dynamic programming, Princeton Univ. Press, Princeton, N. J. (1957).
2) Bellman, R., et al.: Applied Dynamic Programming, Princeton Univ. Press, Princeton, N. J. (1962).
3) Hartmanis, et al.: Classification of computations by time and memory requirements, Proc. IFIP Congress 65, Spartan Books, Washington, DC, pp. 31-35.
4) Edmonds, J.: Paths, trees, and flowers, Canad. J. Math., Vol. 17, pp. 449-467.
5) Tarjan, R. E.: Complexity of combinatorial algorithms, SIAM Review, Vol. 20, No. 3, pp. 457-491 (1977).
6) Jazayeri, M., et al.: The intrinsically exponential complexity of the circularity problem for attribute grammars, CACM, Vol. 18, No. 12, pp. 697-706 (1975).
7) Knuth, D. E.: Semantics for context free languages, Math. Syst. Th. Vol. 2, pp. 127-145 (1968), Math Syst. Th. Vol. 5, p. 95 (1971) に訂正がある．
8) Cook, S. A.: The complexity of theorem proving procedures, Proc. Third Annual ACM Symp. on Theory of Computing, ACM, N. Y., pp. 151-158 (1971).
9) Aho, A. V., et al.: The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass. (1974).
10) Weide, B.: A survey of analysis techniques for discrete algorithms, Comp. Surveys, Vol. 9, No. 4, pp. 291-313 (1977).
11) Knuth, D. E., et al.: Fast pattern matching in strings, SIAM J. Comput. Vol. 6, No. 2, pp. 323-350 (1977).
12) Solovay, R., et al.: A fast Monte-Carlo test for primarity, SIAM J. Comput. Vol. 6, No. 1, pp. 84-85 (1977).

(昭和55年1月8日受付)